VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra telekomunikační techniky

# Analýza T-SQL kódu obsahujícího kurzor

# Analysis of T-SQL Code with Cursor

2018 Matěj Kotyz

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

# Zadání bakalářské práce

Student: **Matěj Kotyz**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Analýza T-SQL kódu obsahujícího kurzor
Analysis of a T-SQL Code with Cursor

Jazyk vypracování: čeština

Zásady pro vypracování:

T-SQL kód nabízí řadu procedurálních rozšíření SQL jako jsou kurzory, cykly a větvení. Nicméně v nezřídka jsou tyto konstrukce používány nesprávně a místo, aby byl problém vyřešen pouze s pomocí SQL, tak je využito procedurální rozšíření, které je pomalejší. Cílem této práce je napsat analyzátor T-SQL kódu, který bude detekovat problematické řešení nějakého problému. Práce se bude zaměřovat na eliminaci kurzoru.

Práce bude probíhat v následujících krocích:
1. Sestavení alespoň deseti vzorových situací s problematickým T-SQL kódem obsahující kurzor.
2. Rešerše metod v dané oblasti.
3. Analýza, návrh a implementace aplikace, která bude provádět analýzu problematického kódu.
4. Důkladné otestování na vzorových situacích a jasné popsání problematických situací, které se nástrojem nepodaří detekovat.

Seznam doporučené odborné literatury:

Podle pokynů vedoucího bakalářské práce.

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. Radim Bača, Ph.D.**

Datum zadání: 01.09.2016

Datum odevzdání: 30.04.2018


doc. Ing. Jan Platoš, Ph.D.
*vedoucí katedry*

prof. Ing. Pavel Brandštetter, CSc.
*děkan fakulty*

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 29. června 2018 .....

Rád bych poděkoval vedoucímu práce Ing. Radimu Bačovi, Ph.D. za jeho pomoc při vytváření této bakalářské práce.

**Abstrakt**

V této bakalářské práci se zabývám analýzou T-SQL kódu s kurzorem, za účelem převodu tohoto kódu na deklarativní SQL dotazy. V analýze se věnuji převážně JMD příkazům. K účelu analýzy mi posloužilo několik vzorových příkladů v T-SQL, které jsem následně převedl do SQL. Součástí této práce je také program, který jsem vytvořil pro převod T-SQL kódu s kurzorem na kód bez kurzoru.

**Klíčová slova**: SQL, T-SQL, kurzor, JMD

**Abstract**

This bachelor thesis deals with the analysis of a T-SQL code with cursor and transformation this code to declarative SQL queries. In analysis I mainly deal with DML statements. In the analysis I used some sample examples in T-SQL, which I then converted to SQL. Part of this bachelor thesis was also the creation of a program that converts T-SQL code with cursor to code without cursor.

**Key Words**: SQL, T-SQL, cursor, DML

# Obsah

# Seznam použitých zkratek a symbolů

| | | |
|---|---|---|
| SQL | – | Structured Query Language |
| T-SQL | – | Transact - Structured Query Language |
| JMD | – | Jazyk pro manipulaci s daty |
| JDD | – | Jazyk pro definici dat |
| DCL | – | Data Control Langage |
| TCL | – | Transaction Control Language |
| ANSI | – | American National Standards Institute |
| ISO | – | International Organization for Standardization |
| DML | – | Data manipulation language |
| XML | – | Extensible Markup Language |

# Seznam obrázků

# Seznam tabulek

# Seznam výpisů zdrojového kódu

# 1 Úvod

V této bakalářské práci se zabývám analýzou T-SQL kódu s kurzorem. V úvodu si nastíníme, proč se touto problematikou vůbec zabýváme. Dále si projdeme teoretickou část, ve které si řekneme něco o SQL, T-SQL a kurzorech. Následně si ukážeme, jak vypadá kurzor v T-SQL a projdeme si definice příkazů, které jsou součástí kurzoru. Poté budeme řešit samotnou analýzu T-SQL kódu s kurzorem. V této analýze se budu věnovat především JMD příkazům. Nejpodrobněji je zde popisován příkaz DELETE, na němž si projdeme nejvíce situací, které mohou nastat. V analýze budeme postupovat od nejjednodušších příkladů T-SQL kódu s kurzorem, které se budeme snažit co nejvíce zobecnit. Analýza končí složitějšími konkrétními příklady, u kterých je také zachycen převod na deklarativní SQL dotazy. Před závěrem si popíšeme desktopovou aplikaci, kterou jsem vytvořil pro tuto bakalářskou práci. Závěrem shrnu vše, co se mi podařilo, a také vše, co by se v budoucnu ještě mohlo vylepšit.

# 2 Úvod do problematiky

V této bakalářské práci se budu zabývat problematikou spjatou s převodem procedurálního T-SQL kódu na deklarativní SQL kód. Abych to ještě přiblížil, bude se jednat o T-SQL kód s kurzorem.

Dříve než přistoupíme k řešení problematiky jako takové, řekneme si, z jakého důvodu se chceme zaměřit na kurzory a jejich převod do klasického SQL. Mnoho programátorů upřednostňuje psaní kódu skrze nějaký procedurální jazyk, protože je to z jejich pohledu jednodušší a komplexní úlohy se jim tak řeší snáz. Naopak řešit nějaký komplexní problém jen skrz SQL dotazy nemusí být zrovna jednoduché a hned na první pohled zřejmé. Z těchto důvodu se v této bakalářské práci budu zabývat popisováním a následným převodem T-SQL kódu do SQL.

Kurzor je ve srovnání s SQL dotazy velice neefektivní nástroj, který se velice často, takřka vždy, využívá na řešení problémů, které jdou vyřešit pomocí klasického SQL. Pokud se určitý problém dá řešit pomoci SQL dotazů, tak jeho vykonání většinou stojí mnohem méně strojového času, než kdyby stejný problém byl řešen kurzorem v T-SQL. Jedná se tedy hlavně o optimalizaci již stávajícího kódu.

Hlavním důvodem, proč je řešení pomocí kurzoru o tolik pomalejší, než při řešení téhož problému jen s SQL dotazem, je ten, že v kurzoru se musí pro každý záznam (řádek) vykonat jeden nebo více klasických SQL dotazů na serveru. Ke každému dotazu se musí vytvořit plán zpracování a režie kolem taky stojí nějaký ten strojový čas. Kdežto když nepoužijeme kurzor, ale použijeme jen kód s SQL dotazem, tak si server udělá plán zpracování nad daným dotazem a vykoná ho pouze jednou a ještě zvolí optimální/nejrychlejší konečné řešení zpracování dotazu a ten se následně provede.

Existuje již mnoho nástrojů pro optimalizaci databázových serverů. Tyto nástroje na základě monitoringu dokáží dělat různé optimalizace, například přidání indexu a jiné. Bohužel i když nějaké nástroje dokáží do jisté míry optimalizovat T-SQL kód, jako jsou například nástroje Qure [A.3], nenašel jsem žádný, který by dokázal převést mé vzorové T-SQL kódy s kurzorem na deklarativní SQL kód.

Abychom si mohli problematiku přiblížit, musíme si nejdříve uvést nějaké základy, na kterých budeme stavět. Pojďme si tedy něco říci o SQL, T-SQL a kurzoru.

## 2.1 SQL

SQL (Structured Query Language) je standardizovaný programovací jazyk, který je zaměřený na práci s daty uloženými v relačních databázích. SQL se využívá hlavně v kontextu spojeným s databázemi, a to ať s jejich vytvořením, úpravou nebo přímo pro práci s daty. Dále SQL obsahuje funkčnost pro bezpečnost, transakce a jiné. Jeho počátky byly v sedmdesátých letech 19. století a od té doby se nadále vyvíjí. Postupem času se objevily oficiální standardy pro SQL. Do dnešní doby se SQL stále vyvíjí a velké organizace jako Microsoft, Oracle a další k němu přidávají svoje rozšíření. SQL příkazy se často dělí do skupin dle jejich funkčnosti. Jedna skupina

příkazu patří pod JMD (jazyk pro manipulaci s daty). Tyto příkazy jsou pro práci přímo s daty, jako například příkazy select, update, delete, insert. Další skupina příkazů je zahrnuta pod JDD (jazyk pro definici dat). Zde jsou příkazy create, alter, drop a jiné, které jsou spjaty s vytvořením a úpravou relační databáze. Dále jsou v SQL i příkazy DCL (Data Control Language), které umožňují přidělit, či odebrat práva. Mimo jiné zde máme i příkazy pro transakce (TCL). [1]

## 2.2 T-SQL

V této bakalářské práci se převážně zabývám T-SQL kódem, tak si pojďme přiblížit, co to vlastně T-SQL je. T-SQL (Transact-Structured Query Language) je procedurální rozšíření standardního SQL od firmy Microsoft a Sybase, které zahrnuje řídící konstrukce jako větvení, cykly, proměnné, výjimky, funkce, různé úpravy stávajících příkazů a mnoho dalších rozšíření. T-SQL se používá pro komunikaci s Microsoft SQL serverem. [2]

## 2.3 T-SQL Kurzor

Kurzor je vlastně datová struktura, která nám umožňuje postupně procházet záznamy v databázi, řádek po řádku. Kurzory jsou spjaty s procedurálním programováním.

### 2.3.1 Definice deklarace kurzoru

V T-SQL můžeme použít dvě formy zápisu deklarace kurzoru, ale jejich parametry není možno kombinovat. První forma je podle normy ISO (viz obr. 1). Druhá forma zápisu vychází přímo z T-SQL a je zachycena na obrázku 2.[3]

```
DECLARE cursor_name [ INSENSITIVE ] [ SCROLL ] CURSOR
     FOR select_statement
     [ FOR { READ ONLY | UPDATE [ OF column_name [ ,...n ] ] } ]
[;]
```

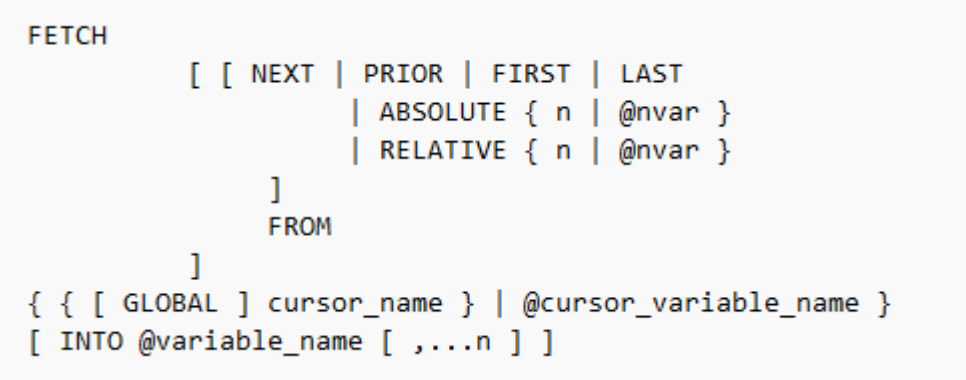
Obrázek 1: Definice kurzoru dle standardu ISO [3]

```
DECLARE cursor_name [ INSENSITIVE ] [ SCROLL ] CURSOR
     FOR select_statement
     [ FOR { READ ONLY | UPDATE [ OF column_name [ ,...n ] ] } ]
[;]
```

Obrázek 2: Definice deklarace kurzoru dle T-SQL [3]

### 2.3.2 Příkaz FETCH

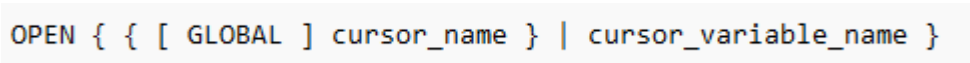
Tento příkaz slouží k načtení konkrétního záznamu (řádku) z kurzoru. Syntaxe příkazu FETCH zachycuje obrázek 3. [4]




Obrázek 3: Definice příkazu FETCH v T-SQL [4]

### 2.3.3 Příkaz Open

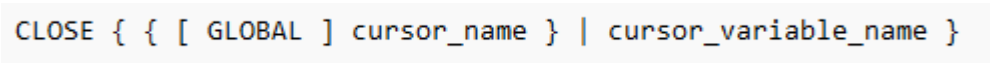
Příkaz OPEN otevře kurzor a naplní jej dle deklarace kurzoru. Syntaxi příkazu vidíme na obrázku 4. [6]




Obrázek 4: Definice příkazu OPEN v T-SQL [6]

### 2.3.4 Příkaz Close

Příkaz CLOSE uzavře otevřený kurzor, uvolní data a zámky na záznamech. Syntaxe příkazu CLOSE je zachycena na obrázku 5. [5]




Obrázek 5: Definice příkazu CLOSE v T-SQL [5]

### 2.3.5 Příkaz Deallocate

Příkaz DEALLOCATE odebere reference na kurzor, jakmile jsou všechny reference odebrány, tak se uvolní i data na serveru. Syntaxi příkazu vidíme na obrázku 6. [7]

### 2.3.6 Cyklus Kurzoru

Poslední část spjata s kurzorem je cyklus, díky kterému můžeme procházet záznam po záznamu. Na ukázce cyklu (viz výpis 1) si můžeme všimnout již dříve zmiňovaného příkazu FETCH, který nám naplní proměnné v T-SQL daty z konkrétního záznamu. Dále zde máme WHILE cyklus s podmínkou na `@@FETCH_STATUS`. `@@FETCH_STATUS` je vlastně funkce, která nám vrací stav naposled vykonaného příkazu FETCH. Tabulka návratových hodnot funkce `@@FETCH_STATUS` nám

```
DEALLOCATE { { [ GLOBAL ] cursor_name } | @cursor_variable_name }
```

Obrázek 6: Definice příkazu CLOSE v T-SQL [7]

řekne, které hodnoty může `@@FETCH_STATUS` nabývat a co ty hodnoty znamenají (viz tabulka 1).

```
(blok kódu)
FETCH :atributy_kurzoru: FROM :jmeno_kurzoru: INTO :seznam_promennych:
WHILE @@FETCH_STATUS = 0
BEGIN
   (blok kódu)
FETCH :atributy_kurzoru: FROM :jmeno_kurzoru: INTO :seznam_promennych:
END
(blok kódu)
```

Výpis 1: Cyklus pro kurzor

Tabulka 1: Tabulka návratových hodnot funkce FETCH STATUS

| **Návratová hodnota funkce** | **popis významu** |
|---|---|
| 0 | příkaz fetch proběhl v pořádku |
| -1 | při příkazu fetch nastala chyba |
| -2 | záznam pro načtení chybí |
| -9 | kurzor neprovádí fetch operaci |

# 3 Analýza T-SQL kódu s kurzorem

Abychom zdárně převedli T-SQL kód s kurzorem do deklarativního SQL kódu, musíme si nejdříve určit, jak vůbec může vypadat kurzor v T-SQL. K tomuto účelu nám poslouží následující upravený T-SQL kód (viz výpis 2), kde si nastíníme jak kurzor v T-SQL vypadá.

```
(blok1)
DECLARE jmeno_kurzoru CURSOR FOR SELECT...
(blok2)
OPEN jmeno_kurzoru;
(blok3)
FETCH NEXT FROM jmeno_kurzoru INTO @promenna1, ...
(blok4)
WHILE @@FETCH_STATUS = 0
(blok5)
BEGIN
    (blok6)
    FETCH NEXT FROM jmeno_kurzoru INTO @promenna1, ...
    (blok7)
END
(blok8)
CLOSE jmeno_kurzoru
(blok9)
DEALLOCATE jmeno_kurzoru
(blok10)
```

Výpis 2: Ukázka cyklu s kurzorem

Ve výpisu 2 můžeme vidět klasickou kostru pro jednoduchý kurzor v T-SQL. Jakýkoliv blok na ukázce může být nahrazen jakýmkoliv množstvím příkazů z T-SQL. Abychom mohli korektně převést T-SQL kód s kurzorem na SQL dotaz, musíme brát v potaz všechny možnosti, které mohou nastat. Jelikož v jakémkoliv bloku může být nekonečně mnoho příkazů, bude to nelehký oříšek. I když počet příkazů jazyka T-SQL a všech jejich variant je konečný, jejich kombinací může být nespočetně.

Na následujících stránkách systematicky projdeme situace, ke kterým může dojít. K popisu těchto situací budu používat převážně příkaz DELETE a u ostatních JMD příkazů již popíši jen jejich odlišnosti. Následně si uvedeme několik konkrétních příkladů.

## 3.1 Používané termíny v analýze

Než se vrhneme do analýzy, uvedeme si pár pojmů, se kterými budeme nadále pracovat.

**vázaná proměnná**

Pro tuto bakalářskou práci budeme vázanou proměnnou chápat, jako proměnnou, u které se hodnota v těle cyklu kurzoru mění, pouze skrz FETCH příkaz.

**konstantní proměnná**

Pro tuto bakalářskou práci budeme konstantní proměnnou chápat, jako proměnnou, u které se hodnota v těle cyklu kurzoru nemění.

**proměnná set**

Pro tuto bakalářskou práci budeme proměnnou set chápat, jako proměnnou, u které se hodnota v těle cyklu kurzoru mění jinak, než skrz FETCH příkaz.

## 3.2 DELETE

Zde si uvedeme příklad T-SQL kódu s kurzorem (viz výpis 3), který nám poslouží jako referenční. Tento referenční příklad budeme hojně využívat a to tak, že v něm budeme postupně zcela měnit příkazy v těle cyklu. Jinak řečeno komentář č.1 a komentář č.2 nám vymezuje prostor, který se bude měnit. Tento referenční příklad budeme postupně upravovat a rozšiřovat o další příkazy, abychom si prošli situace, které mohou nastat.

```
DECLARE cursor_tab1 CURSOR FOR SELECT ID,firstname FROM [Customer2]
DECLARE @id_tab1 INT
DECLARE @name as NVARCHAR(50);
DECLARE @variable INT
SET @variable = 5
OPEN cursor_tab1
FETCH NEXT FROM cursor_tab1 INTO @id_tab1, @name
WHILE @@FETCH_STATUS = 0
BEGIN
   /*komentář č.1 */
   DELETE FROM [Customer]
   /*komentář č.2 */
FETCH NEXT FROM cursor_tab1 INTO @id_tab1, @name
END
CLOSE cursor_tab1
DEALLOCATE cursor_tab1
```

Výpis 3: Referenční příklad T-SQL kódu s kurzorem

Jelikož je tento příklad první, rozebereme si jej podrobněji. V této ukázce T-SQL kódu máme jako první příkaz pro deklaraci kurzoru. Z deklarace kurzoru můžeme vyčíst jméno kurzoru a SELECT výraz, který pro nás bude velmi důležitý. Dále následuje příkaz SET, který slouží k přiřazení výrazu na pravé straně k proměnné na levé straně. Jako další příkaz je zde příkaz pro otevření kurzoru, ve kterém se jen specifikuje, o který kurzor jde. Na řadu přichází příkaz FETCH, ve kterém se specifikuje jméno kurzoru a proměnné, do kterých se ukládají data z kurzoru. Tento řádek kódu je pro nás velice důležitý. Sděluje nám informaci o tom, jak se budou jednotlivé atributy z deklarace kurzoru mapovat na proměnné (dále jen vázané proměnné). Spárované proměnné s atributy z kurzoru máme zachycené v tabulce 2. Následuje cyklus pro kurzor. V tomto cyklu se nachází příkaz DELETE a příkaz FETCH, který je stejný jako příkaz FETCH před cyklem. Poslední příkazy jsou CLOSE a DEALLOCATE, ve kterých se jen objevuje jméno kurzoru.

Vraťme se na začátek a všimněme si, že v SELECT výrazu u deklarace kurzoru není přítomná klausule ORDER BY, což znamená, že procházení kurzoru bude nedeterministické. Jinak řečeno,

Tabulka 2: Tabulka vázaných proměnných s atributy kurzoru

| Atribut | Proměnná |
|---|---|
| ID | `@id_tab1` |
| firstname | `@name` |

nevíme v jakém pořadí se budou záznamy procházet. Tuto informaci musíme mít na paměti a počítat s ní dále.

Prošli jsme si všechny příkazy z příkladu a nyní se podívejme na to hlavní a to na příkaz DELETE. V tomto případě máme příkaz DELETE bez jakéhokoliv závislosti na kurzoru a díky tomu je kurzor naprosto zbytečný a zapříčiní jen to, že se tento příkaz provede vícekrát. To znamená, že poprvé se smažou všechny záznamy z tabulky a u dalších průchodů cyklem kurzoru se nestane nic, jelikož jsou záznamy již smazány.

K tomu, abychom mohli tvrdit, že korektním převedením toho procedurálního T-SQL kódu na deklarativní SQL, je samotný DELETE dotaz, musíme ještě brát v potaz, zdali není množina záznamů, která je vybrána v deklaraci kurzoru, prázdná. Pokud je prázdná, DELETE dotaz se vůbec nevykoná. K ošetření této možnosti použijeme EXISTS, do kterého vložíme příkaz SELECT z deklarace kurzoru. Řešení je zachyceno ve výpisu 4.

```
DELETE
FROM [Customer]
WHERE EXISTS(
        SELECT ID,firstname
        FROM [Customer2]
        )
```

Výpis 4: Řešení referenčního příkladu

## 3.3 Klauzule WHERE u příkazu DELETE

Pojďme si ukázat, které situace mohou nastat, když k se v referenčním příkladu(viz výpis 3) objeví příkaz DELETE s klauzulí WHERE.

### 3.3.1 Predikát s konstantou

Příklad s konstantou (viz výpis 6) má řešení zachyceno ve výpisu 7.

Postup při řešení příkladu je jednoduchý, neuděláme zde nic jiného, než že stávající výraz za klauzulí WHERE dáme do závorek, přidáme logickou spojku AND a přidáme EXISTS pro ověření, zdali nám příkaz SELECT z deklarace kurzoru vrací nějaký záznam. Stejně budeme postupovat, když se v příkladu místo konstanty objeví proměnná „@variable“. Jelikož se jedná o konstantní proměnnou(tuto proměnnou jsme si popsali na začátku v sekci 3.1).

Obdobně se řeší i příklady, ve který se v klauzuli WHERE u příkazu DELETE objeví výraz, který může být vygenerován dle pravidel gramatiky zachycených ve výpisu 5.

Např. výraz uvID>5 AND firstname='karel', je vytvořen pomoci uvedených pravidel a budeme ho tedy moci převést stejně.

```
V → V AND V | V OR V | NOT V | P
P → a > k | a < k | a>=k | a<=k


Kde
a je atribut(sloupec tabulky)
k je konstanta nebo konstantní proměnná
```

Výpis 5: Pravidla pro predikát s konstantou

```
DELETE
FROM [Customer]
WHERE ID=1;
```

Výpis 6: Příklad s konstantou

```
DELETE FROM [Customer]
WHERE (ID=1) AND EXISTS(
                    SELECT ID,firstname
                    FROM [Customer2]
                    )
```

Výpis 7: Řešení příkladu s konstantou

### 3.3.2 Predikát s vázanou proměnnou

Příklady s vázanou proměnnou se budou muset řešit individuálně dle operátoru použitého v predikátu podmínky WHERE. Pro popis příkladů budeme stále používat referenční příklad (viz výpis 3).

#### 3.3.2.1 Operátor rovná se

Na příkladu zachyceném ve výpisu 8 vidíme, že se zde v klauzuli WHERE objevila vázaná proměnná „`@id_tab1`“. Tuto proměnnou máme spárovanou s atributem („ID“) z deklarace kurzoru. Spárované proměnné máme zachyceny v tabulce 2.

```
DELETE FROM [Customer]
WHERE ID=@id_tab1
```

Výpis 8: Příklad s proměnnou vázanou na kurzor

```
DELETE gnt1
FROM [Customer] gnt1
JOIN(
      SELECT ID,firstname
      FROM [Customer2]
      ) gnt2 On gnt2.ID=gnt1.ID
```

```
WHERE EXISTS(
            SELECT ID,firstname
            FROM [Customer2]
            )
```

Výpis 9: Řešení příkladu s proměnnou vázanou na kurzor

#### 3.3.2.2 Operátory <,>,>=,<=

```
DELETE
FROM [Customer]
WHERE ID<@id_tab1
```

Výpis 10: Příklad s proměnnou vázanou na kurzor

V příkladu (viz výpis 10) se nám vyskytla vázaná proměnná. Tuto informaci musíme brát v potaz a k řešení si musíme vyjasnit, co nám tento T-SQL kód vlastně dělá. V každém průchodu cyklu se nám smaže množina záznamů z tabulky v závislosti na aktuální hodnotě proměnné. Ve skutečnosti nás zajímá jen průchod, ve kterém je hodnota proměnné nejvyšší, jelikož se v tomto průchodu smaže nejvíce záznamů a ostatní průchody jsou již nadbytečné, neboť mažou jen záznamy, které jsou už jen podmnožinou smazaných záznamů. K řešení využijeme funkci MAX. Řešení je zachyceno ve výpisu 11.

```
DELETE gnt1
FROM [Customer] gnt1
WHERE gnt1.ID < (
                SELECT MAX(ID)
                FROM Customer2
                )
AND
EXISTS(SELECT ID,firstname FROM [Customer2])
```

Výpis 11: Příklad s proměnnou vázanou na kurzor

Obdobně by řešení vypadalo i pro operátor <=, kde se ve výsledku změní jen operátor.

Pro operátor > a >= by by řešení vypadalo obdobně, jen by se v řešení nepoužila funkce MAX, ale použila by se funkce MIN.

#### 3.3.2.3 Kombinace operátorů a logických spojek

Zde si uvedeme několik příkladu, ve kterých se objevují různé kombinace operátorů a logických spojek.

```
WHERE firstname=@name AND ID>@id_tab1
```

Výpis 12: Příklad pro operátor AND

K řešení příkladu 12 využijeme příklady, které jsme si již popisovali. Jednotlivé podmínky v klauzuli WHERE jsme již řešili a tak si pojďme hned ukázat řešení, které je zachyceno ve výpisu 13.

```
DELETE gnt1
FROM [Customer] gnt1
Join(
      SELECT ID,firstname
      FROM [Customer2]
      ) gnt2 On gnt2. firstname = gnt1. firstname
WHERE EXISTS(
                SELECT ID,firstname
                FROM [Customer2]
                )
AND
gnt1.ID > (
            SELECT MIN(ID)
            FROM Customer2)
AND
EXISTS(
      SELECT ID,firstname
      FROM [Customer2]
      )
```

Výpis 13: Řešení příkladu pro operátor AND

```
WHERE ID=@id_tab1 OR ID>@id_tab1
```

Výpis 14: Další s proměnnou pro operátor OR

Ve výpisu 14 se nám v první i druhé podmínce vyskytla vázaná proměnná. Jelikož jsme si řešení daných situací již výše prošli, tak se pojďme podívat hned na výsledek, který je zachycený ve výpisu 15.

```
DELETE gnt1 FROM [Customer] gnt1
Join(
      SELECT ID,firstname
      FROM [Customer2]
      ) gnt2
On gnt2.ID=gnt1.ID OR gnt1.ID > (
                                  SELECT MIN(ID)
                                  From Customer2
                                  )
WHERE EXISTS(
                SELECT ID,firstname
                FROM [Customer2]
                )
```

Výpis 15: Řešení příkladu pro operátor OR

### 3.3.3 Predikát s proměnnou set

Příklady s proměnnou set se budou muset řešit individuálně, jelikož je není tak jednoduché zobecnit, jako příklady s konstantou, či vázanou proměnnou. Nějaké příklady řešení jsou zachyceny v sekci 3.10 konkrétních příkladů.

## 3.4 Mnohonásobný výskyt JMD příkazu v jednom cyklu kurzoru

Nyní se podíváme na situaci, kdy se v jednom cyklu kurzoru objeví dva DELETE příkazy.

```
DELETE
FROM [Customer]
WHERE ID = 1

DELETE FROM [Customer2]
WHERE ID = @id_tab1
```

Výpis 16: Příklad pro mnohonásobný výskyt JMD příkazu v cyklu kurzoru

Jako příklad použijeme referenční příklad (3) s tělem cyklu, který je zachycený ve výpisu 16. Jelikož se v cyklu objevili dva DELETE příkazy, tak po převodu na kód bez kurzoru budou také dva DELETE příkazy. Budeme postupovat tak, že nejdříve převedeme první příkaz DELETE podle pravidel, které jsme si již popsali výše a poté převedeme druhý příkaz DELETE. Řešení můžeme vidět ve výpisu 17.

```
DELETE
FROM [Customer]
WHERE (ID=1) AND EXISTS(
                        SELECT ID,firstname
                        FROM [Customer2]
                        )

DELETE gnt1 FROM [Customer] gnt1
JOIN(
     SELECT ID,firstname
     FROM [Customer2]
     ) gnt2
On gnt2.ID=gnt1.ID
WHERE EXISTS(
             SELECT ID,firstname
             FROM [Customer2]
             )
```

Výpis 17: Řešení příkladu pro mnohonásobný výskyt JMD příkazu v cyklu kurzoru

Zde u tohoto příkladu k žádnému konfliktu nedojde, jelikož oba příkazy jsou typu DELETE a mažou záznamy z různých tabulek. K zamyšlení ale je, zdali bude takto dosažený výsledek korektní, i kdyby první příkaz byl UPDATE či INSERT a druhý DELETE nad stejnou tabulkou. Tuto úvahu necháme na později a vrhneme se na příklady s podmínkou IF.

## 3.5 Podmínka IF s vnořeným DELETE příkazem

Nyní si ukážeme, co se stane, když přidáme podmínku „IF“ do těla cyklu.

```
V → V AND V | V OR V | NOT V | P
P → v>k | k>v | v<k | k<b | v=k | k=v | v>=k | k>=v | v<=k | k<=v | v>v | v<v | v=v
    | v>=v | v<=v
```

```
Kde
v je vázaná proměnná
k je konstanta nebo konstantní proměnná
```

Výpis 18: Pravidla pro if s konstantou a vázanou proměnnou

Pokud se nám v podmínce IF vyskytne výraz Y, který může být vytvořen podle pravidel zachycených ve výpisu 18, tak tuto podmínku můžeme přesunout do klauzule WHERE z příkazu SELECT u deklarace kurzoru. Pokud již v klauzuli WHERE byl nějaký výraz, tak jej dáme do závorek a přidáme logickou spojku AND mezi starý výraz a přesunutý výraz z podmínky IF. Při přesunu výrazu z podmínky IF do příkazu SELECT z deklarace kurzoru musíme ještě zaměnit vázané proměnné za atributy, na které jsou vázané.

Jako příklad použijeme referenční příklad (3) s tělem cyklu, který je zachycený ve výpisu 19. Řešení takového příkladu můžeme vidět ve výpisu 20.

```
IF @id_tab1 >15
   DELETE
   FROM [Customer]
   WHERE ID=@id_tab1
```

Výpis 19: Příklad s podmínkou IF

```
DELETE gnt1
FROM [Customer] gnt1
JOIN(
      SELECT ID,firstname
      FROM [Customer2]
      WHERE ID=15) gnt2
On gnt2.ID=gnt1.ID
WHERE EXISTS(
               SELECT ID,firstname
               FROM [Customer2]
               WHERE ID=15
               )
```

Výpis 20: Řešení příkladu č.1 s podmínkou IF

Pokud se nad tímto příkladem trošku zamyslíme, tak korektním řešením je také řešení, kdy změníme klauzuli WHERE u příkazu DELETE na porovnání přímo s danou konstantou a řešení by tedy bylo následující (viz výpis 21).

```
DELETE
FROM [Customer]
WHERE ID=15 AND EXISTS(
                        SELECT ID,firstname
                        FROM [Customer2]
                        where ID=15
                        )
```

Výpis 21: Lepší řešení příkladu č.1 s podmínkou IF

### 3.5.1 Shrnutí pro IF

Pokud se v podmínce objeví vázaná proměnná, konstanta nebo konstantni proměnná a logické nebo porovnávací operátory, tak podmínku lze přesunout do SELECT výrazu z deklarace kurzoru.

Další zajímavosti je, že pokud je v podmínce operátor = a na jedné straně vázaná proměnná a konstanta nebo konstantní proměnná na straně druhé a zároveň je tato vázaná proměnná přítomná v příkazu DELETE za WHERE klauzuli, tak tuto spárovanou proměnnou ve WHERE klauzuli, můžeme nahradit konstantou a řešení se nám výrazně zjednoduší.

## 3.6 Podmínka IF s ELSE

Podívejme se teď na příklad s IF, který má i ELSE větev.

```
IF @id_tab1 >15
   DELETE
   FROM [Customer]
   WHERE ID=@id_tab1
ELSE
   DELETE
   FROM [Customer]
   WHERE firstname= 'karel'
```

Výpis 22: Příklad s podmínkou IF a ELSE větví

Jako příklad použijeme referenční příklad (3) s tělem cyklu, který je zachycený ve výpisu 22. Jelikož jsou zde dva příkazy DELETE, tak výsledkem budou také dva příkazy DELETE. K žádnému konfliktu zde nedojde i kdyby se jednalo o jiné JMD příkazy, protože se ve skutečnosti pro každý průchod cyklem kurzoru stane aktivní jen jeden příkaz DELETE. Nejdříve převedeme příkaz DELETE z kladné větve podmínky, následně převedeme příkaz DELETE z ELSE větve. ELSE větev převádíme tak, že podmínku znegujeme a dále postupujeme naprosto stejně. Výsledný převod můžeme vidět na výpisu 23.

```
DELETE gnt1
FROM [Customer] gnt1
JOIN(
      SELECT ID,firstname
      FROM [Customer2]
      WHERE ID>15
      ) gnt2 On gnt2.ID=gnt1.ID
WHERE EXISTS(
                SELECT ID,firstname
                FROM [Customer2]
                WHERE ID>15
                )

DELETE FROM [Customer]
WHERE firstname= 'karel' AND EXISTS(
                                        SELECT ID,firstname
                                        FROM [Customer2]
```

```
WHERE NOT(ID>15)
)
```

Výpis 23: Příklad s podmínkou IF a ELSE větví

## 3.7 Klauzule SET u příkazu UPDATE

Příkaz UPDATE budeme řešit obdobně jako DELETE, důležitý rozdíl je ale ten, že se zde navíc objevuje klauzule SET.

### 3.7.1 Přiřazení konstanty

Jako příklad použijeme referenční příklad (3) s tělem cyklu, který je zachycený ve výpisu 24. Příkaz DELETE z referenčního příkladu (viz výpis 3) zaměníme za následující příkaz UPDATE.

```
UPDATE [Customer]
SET firstname='text'
WHERE ID=@id_tab1
```

Výpis 24: Příklad pro UPDATE s přiřazením konstanty

Postup řešení bude obdobný jako u příkazu DELETE. Důležitá je zde klauzule SET, ve které se nám nyní objevilo přiřazení s konstantou. U přiřazení konstanty se nám žádné omezení nevyskytuje, a tak příklad můžeme jednoduše převést do SQL dotazu (viz výpis 25).

```
UPDATE gnt1
SET firstname = 'text'
FROM Customer gnt1
JOIN (
        SELECT ID,firstname
        FROM [Customer2]
        ) gnt2 ON gnt2.ID = gnt1.ID
WHERE EXISTS(
                SELECT ID,firstname
                FROM [Customer2]
                )
```

Výpis 25: Řešení příkladu pro UPDATE s přiřazením konstanty

### 3.7.2 Přiřazení konstanty ke stávající hodnotě atributu

```
UPDATE [Customer]
SET firstname= firstname +'text'
WHERE ID=@id_tab1
```

Výpis 26: Příklad pro UPDATE s přiřazením konstanty ke stávající hodnotě atributu

Zde jako příklad použijeme referenční příklad s tělem cyklu, který je zachycený ve výpisu 26. Postup řešení bude obdobný jako u příkazu DELETE. Důležitá je zde klauzule SET, ve které

se nám nyní objevilo přiřazení s konstantou a atributem dané tabulky. Ani v tomto případě při převodu nenastane žádný problém. Řešení příkladu je zachyceno ve výpisu 27.

```
UPDATE gnt1
SET firstname = gnt1.firstname+'text'
FROM Customer gnt1
JOIN (
        SELECT ID,firstname
        FROM [Customer2]
        ) gnt2 ON gnt2.ID = gnt1.ID
WHERE EXISTS(
                SELECT ID,firstname
                FROM [Customer2]
                )
```

Výpis 27: Řešení příkladu pro UPDATE s přiřazením konstanty ke stávající hodnotě atributu

### 3.7.3 Přiřazení vázané proměnné

```
UPDATE [Customer]
SET firstname= @name
WHERE ID=@id_tab1
```

Výpis 28: Příklad pro UPDATE s přiřazením vázané proměnné

Jako příklad použijeme referenční příklad (3) s tělem cyklu, který je zachycený ve výpisu 28. Postup řešení bude obdobný jako u příkazu DELETE. Důležitá je zde klauzule SET, ve které se nám nyní objevilo přiřazení s vázanou proměnnou. V tomto případě musíme brát v potaz to, že v deklaraci kurzoru není přítomná klauzule ORDER BY. To pro nás znamená, že nevíme, v jakém pořadí se bude UPDATE záznamů provádět. Problém nastane tehdy, když v tabulce „Customer2“ je více záznamů se stejným „ID“. Aby následující řešení bylo korektní, potřebujeme, aby „ID“ z „Customer2“ byl při nejmenším unikátní atribut(např. primární klíč).

Řešení příkladu, pokud je atribut „ID“ z „Customer2“ unikátní, je zachyceno na výpisu 29.

```
UPDATE gnt1
SET firstname = gnt2.firstname
FROM Customer gnt1
JOIN (
        SELECT ID,firstname
        FROM [Customer2]
        ) gnt2 ON gnt2.ID = gnt1.ID
WHERE EXISTS(
                SELECT ID,firstname
                FROM [Customer2]
                )
```

Výpis 29: Řešení příkladu pro UPDATE s přiřazením vázané proměnné

## 3.8 Klauzule VALUES u příkazu INSERT INTO

Příkaz INSERT INTO budeme řešit stejně jako DELETE. Důležitý rozdíl je ale ten, že je zde navíc klauzule VALUES.

Jako příklad použijeme referenční příklad (3) s tělem cyklu, který je zachycený ve výpisu 30.

```
INSERT INTO Customer(ID,firstname,city)
VALUES (@id_tab1, @variable, 'mesto')
```

Výpis 30: Příklad pro INSERT INTO

V příkladu se nám v klauzuli VALUES objevuje vázaná proměnná, konstantní proměnná a konstanta. Řešení příkladu je zachyceno na výpisu 31.

```
DECLARE @variable INT
Set @variable = 5
INSERT INTO Customer(ID,firstname,city)
SELECT ID, @variable, 'mesto'
FROM tab1
WHERE EXISTS(
            SELECT ID,firstname
            FROM Customer
            )
```

Výpis 31: Řešení příkladu pro INSERT INTO

V dalším kroku můžeme odstranit proměnnou „@variable“ a zaměnit ji za její hodnotu.

## 3.9 Shrnutí JMD příkazů

Zde si uděláme shrnutí příkladů, které jsme si doposud prošli.

T-SQL kód s kurzorem můžeme převést na T-SQL kód bez kurzoru za splnění všech těchto podmínek:

- Pokud se v těle cyklu kurzoru objeví pouze JMD příkazy DELETE.
- Příkazy DELETE v těle cyklu kurzoru ve své klauzuli WHERE mohou obsahovat pouze vázané proměnné, konstantní proměnní a konstanty(samozřejmě i logické a porovnávací operátory).
- Podmínky IF, jestli jsou přítomny, tak mohou obsahovat pouze vázané proměnné, konstantní proměnné a konstanty(samozřejmě i logické a porovnávací operátory).

Dále se cyklu kurzoru můžeme zbavit, pokud se v těle cyklu kurzoru objeví pouze jeden JMD příkaz UPDATE a v klauzuli SET nebo WHERE se objeví pouze vázané proměnné, konstantní proměnné nebo konstanty. Je zde ještě jedno omezení a to, že když se v klauzuli SET objeví přiřazení s vázanou proměnnou, tak v podmínce WHERE u daného UPDATE příkazu se nesmí objevit vázaná proměnná, u které atribut, na který je vázaná není unikátní (tzn. pokud je daný atribut unikátní, tak převod do SQL je možné provést korektně). Jinak řečeno, pokud by tato

unikátnost nebyla zaručena, tak by se pro jeden záznam udělalo více aktualizací hodnot a my nevíme, která z hodnot v daném atributu by se nakonec objevila.

Dále se cyklu kurzoru můžeme zbavit, pokud se v těle cyklu kurzoru objeví pouze jeden JMD příkaz INSERT INTO a ten nemá přítomnu klauzuli WHERE nebo je v ní jen konstanta, či konstantní proměnná, a v klauzuli VALUES se objeví vázaná proměnná, konstantní proměnná, konstanta nebo jejich kombinace.

V budoucnu je ještě nutno zanalyzovat kombinace JMD příkazů v jednom cyklu kurzoru. Dále by bylo vhodné zanalyzovat operátor OR v klauzuli WHERE u jiných příkazů než DELETE.

Nyní se vrhneme na konkrétní příklady, ve kterých si ukážeme i případy, které jsme doposud ještě neřešili.

## 3.10 Konkrétní příklady

### 3.10.1 Společné podmínky pro všechny konkrétní příklady

Nutné podmínky, které musí být u příkladů splněny, aby jejich následující konverze byla korektní:

- Pokud neznáme strukturu databáze, tak v deklaraci kurzoru musí být uvedeny atributy u klauzule SELECT, aneb nemůže zde být použita tečková notace, jelikož jen z uvedeného T-SQL kódu neznáme strukturu databáze.
- Všechny příkazy FETCH pro tentýž kurzor musí být totožné.

### 3.10.2 Konkrétní příklad č.1

Tento kus T-SQL kódu(viz výpis32) nedělá nic jiného než to, že změní hodnotu atributu „firstname“ v tabulce „Customer“ pro všechny záznamy, které mají v atributu „city“ hodnotu „Ostrava“. Změna se projeví tak, že se jen ke stávající hodnotě atributu „firstname“ přidá textový řetězec „text“.

```
DECLARE cursor_tab1 CURSOR FOR SELECT ID,firstname
                                    FROM [Customer]
                                    WHERE city='Ostrava'
DECLARE @name as NVARCHAR(50);
DECLARE @id_tab1 as INT;
OPEN cursor_tab1;
FETCH NEXT FROM cursor_tab1 INTO @id_tab1, @name

WHILE @@FETCH_STATUS = 0
BEGIN
   UPDATE [Customer]
   SET firstname= @name + 'text'
   WHERE ID=@id_tab1
FETCH NEXT FROM cursor_tab1 INTO @id_tab1, @name
END
CLOSE cursor_tab1
DEALLOCATE cursor_tab1
```

Výpis 32: Konkrétní příklad č.1

Tabulka 3: Tabulka spárovaných proměnných ke konkrétnímu příkladu č.1

| Atribut | Proměnná |
|---|---|
| ID | `@id_tab1` |
| firstname | `@name` |

Nutné podmínky, které musí být u nynějšího příkladu splněny, aby následující konverze do SQL byla korektní:

- U příkazu UPDATE za klauzulí WHERE musí být přítomná podmínka s vázanou proměnnou.
- U příkazu UPDATE za klauzulí SET se nesmí objevit proměnná set.
- Jak jsme si popsali již v analýze výše, tak řešení uvedené níže (viz výpis 33) je korektní, jen pokud atribut „ID“ z tabulky „Customer“ bude unikátní (např. primární klíč).

Výsledek převodu takového T-SQL kódu na klasické (v angličtině bychom mohli využít výraz „set-based“) SQL by mohl vypadat jako na následující ukázce kódu (viz výpis 33).

```
UPDATE c
SET firstname = t.firstname+'text'
FROM Customer c
JOIN (
  SELECT ID,firstname
  FROM [Customer]
  WHERE city='Ostrava'
) t  ON t.ID = c.ID
WHERE EXISTS(
                SELECT ID,firstname
                FROM [Customer]
                WHERE city='Ostrava'
                )
```

Výpis 33: Řešení konkrétního příkladu č.1

Příklad jsme zdárně převedli, jako další zajímavost v této problematice bych rád ještě nastínil, že každý T-SQL kód, pokud jde převést na SQL, se může zapsat různými způsoby. Stejně to platí i obráceně, že jeden SQL dotaz se může zapsat různě pomocí T-SQL, ale to většinou právě kvůli optimalizaci nechceme. Jako příklad zde uvedu další možný převod do SQL(34), který je možný k předešlému T-SQL kódu.

```
UPDATE [Customer]
SET firstname = firstname+'text'
WHERE ID IN (
```

```
                SELECT ID
                FROM [Customer]
                WHERE city='Ostrava'
                )
AND
EXISTS(
        SELECT ID,firstname
        FROM [Customer]
        WHERE city='Ostrava'
        )
```

Výpis 34: Jíné řešení konkrétního příkladu č.1

### 3.10.3 Konkrétní příklad č.2

Následující T-SQL kód(viz výpis 35) nám vymaže všechny záznamy z tabulky „Customer“, které mají hodnoty atributu „ID“ vyšší než 15.

```
DECLARE @name as NVARCHAR(50);
DECLARE @id_tab1 as INT;
DECLARE cursor_tab1 CURSOR FOR SELECT ID,firstname FROM [Customer]
OPEN cursor_tab1
FETCH NEXT FROM cursor_tab1 INTO @id_tab1, @name

WHILE @@FETCH_STATUS = 0
BEGIN
   IF @id_tab1 >15
      DELETE FROM [Customer]
      WHERE ID=@id_tab1
FETCH NEXT FROM cursor_tab1 INTO @id_tab1, @name
END
CLOSE cursor_tab1
DEALLOCATE cursor_tab1
```

Výpis 35: Konkrétní příklad č.2

Tabulka 4: Tabulka spárovaných proměnných ke konkrétnímu příkladu č.2

| **Atribut** | **Proměnná** |
|---|---|
| ID | `@id_tab1` |
| firstname | `@name` |

Nutné podmínky, které musí být u nynějšího příkladu splněny, aby následující konverze do SQL byla korektní:

- U příkazu DELETE za klauzulí WHERE musí být přítomná podmínka s vázanou proměnnou.
- Za podmínkou IF se může vyskytnout proměnná, která je namapována na kurzor.

Řešení je zachyceno na výpisu 36.

```
DELETE c
FROM [Customer] c
JOIN (
      SELECT ID,firstname
      FROM [Customer]
      WHERE ID>15
      ) t  ON t.ID = c.ID
WHERE EXISTS(
            SELECT ID,firstname
            FROM [Customer]
            WHERE ID>15
            )
```

Výpis 36: Řešení konkrétního příkladu č.2

### 3.10.4 Konkrétní příklad č.3

I když v tomto příkladu (viz výpis 37) se nám vyskytují dva kurzory. Může se nám zdát, že se zde provádí nějaká složitá úprava, opak je pravdou. Jde jen o jednoduché spojení tabulek, které použijeme ke změně atributu „firstname“ u tabulky „tab2“.

```
DECLARE cursor_tab1 CURSOR FOR SELECT ID, firstname FROM tab1
DECLARE cursor_tab2 CURSOR FOR SELECT ID, firstname FROM tab2
DECLARE @name as NVARCHAR(50);
DECLARE @id_tab1 as INT;
DECLARE @name2 as NVARCHAR(50);
DECLARE @id_tab2 as INT;

OPEN cursor_tab1
FETCH NEXT FROM cursor_tab1 INTO @id_tab1, @name
WHILE @@FETCH_STATUS = 0
BEGIN
   OPEN cursor_tab2
FETCH NEXT FROM cursor_tab2 INTO @id_tab2, @name2
WHILE @@FETCH_STATUS = 0
BEGIN
      IF @id_tab2 = @id_tab1
         UPDATE tab2
         SET firstname =@name
         WHERE ID=@id_tab1
      FETCH NEXT FROM cursor_tab2 INTO @id_tab2, @name2
   END
   CLOSE cursor_tab2
DEALLOCATE cursor_tab2
FETCH NEXT FROM cursor_tab1 INTO @id_tab1, @name
END
CLOSE cursor_tab1
DEALLOCATE cursor_tab1
```

Výpis 37: Konkrétní příklad č.3

Nutné podmínky, které musí být u nynějšího příkladu splněny, aby následující konverze do SQL byla korektní:

- U příkazu UPDATE za klauzulí WHERE musí být přítomná podmínka s vázanou proměnnou.
- U příkazu UPDATE za klauzulí SET se nesmí objevit proměnná set.
- Za podmínkou IF se musí vyskytnout proměnné, které jsou namapovány na oba kurzory, aby nám zajistily spojení tabulek.
- Jak jsme si popsali již v analýze výše, tak řešení uvedené níže (viz výpis 41) je korektní, jen pokud atribut „ID“ z tabulky „tab1“ bude unikátní (např. primární klíč).

Příklad zachycený ve výpisu (viz výpis 32) je specifický tím, že je zde jeden kurzor vnořený do druhého. Ke zdárnému vyřešení tohoto příkladu budeme postupovat systematicky. Nejprve vyřešíme vnitřní kurzor a až poté vnější. Řešením celého tohoto příkladu pak bude řešení vnějšího kurzoru.

#### 3.10.4.1 Vnitřní kurzor

Nejprve tedy řešíme vnitřní kurzor (viz výpis 32).

```
...
FETCH NEXT FROM cursor_tab2 INTO @id_tab2, @name2
WHILE @@FETCH_STATUS = 0
BEGIN
      IF @id_tab2 = @id_tab1
         UPDATE tab2
         SET firstname =@name
         WHERE ID=@id_tab1
      FETCH NEXT FROM cursor_tab2 INTO @id_tab2, @name2
   END
   CLOSE cursor_tab2
...
```

Výpis 38: Vnitřní kurzor u konkrétního příkladu č.3

Tabulka 5: Tabulka spárovaných proměnných pro vnitřní kurzor k tabulce tab2

| Atribut | Proměnná |
|---|---|
| ID | `@id_tab2` |
| firstname | `@name2` |

Řešení vnitřního kurzoru máme zachyceno ve výpisu (viz výpis 39).

```
UPDATE tab2
SET firstname =@name
Where ID=@id_tab1 AND EXISTS(
                              SELECT ID, firstname
                              FROM tab2
                              WHERE ID=@id_tab1
                              )
```

Výpis 39: Řešení vnitřního kurzoru u konkrétního příkladu č.3

#### 3.10.4.2 Vnější kurzor

Následně řešíme vnější kurzor (viz výpis 40), kde místo vnitřního kurzoru již máme jeho řešení.

```
...
OPEN cursor_tab1
FETCH NEXT FROM cursor_tab1 INTO @id_tab1, @name
WHILE @@FETCH_STATUS = 0
BEGIN
   UPDATE tab2
   SET firstname =@name
   Where ID=@id_tab1 AND EXISTS(
                                    SELECT ID, firstname
                                    FROM tab2
                                    WHERE ID=@id_tab1
                                    )
FETCH NEXT FROM cursor_tab1 INTO @id_tab1, @name
END
...
```

Výpis 40: Vnější kurzor u konkrétního příkladu č.3

Tabulka 6: Tabulka spárovaných proměnných pro vnější kurzor k tabulce tab1

| Atribut | Proměnná |
|---|---|
| ID | `@id_tab1` |
| firstname | `@name` |

Ve výpisu 41 vidíme řešení vnějšího kurzoru. Toto řešení je také řešením celého příkladu.

```
UPDATE gnt1
SET firstname = gnt2.firstname
FROM tab2 gnt1
JOIN (
      SELECT ID,firstname
      FROM tab1
) gnt2 ON gnt1.ID = gnt2.ID
WHERE EXISTS(
            SELECT ID, firstname
            FROM tab2
            WHERE ID = gnt2.ID)
 AND
 EXISTS(
        SELECT ID,firstname
        FROM tab1
        )
```

Výpis 41: Řešení vnějšího kurzoru u konkrétního příkladu č.3

### 3.10.5 Konkrétní příklad č.4

Následující T-SQL kód (viz výpis 42) nám přidává postupně každý záznam z tabulky „tab1“ do tabulky „tab2“.

```
DECLARE cursor_tab1 CURSOR FOR SELECT ID,firstname FROM tab1
DECLARE @name as NVARCHAR(50);
DECLARE @id_tab1 as INT;
OPEN cursor_tab1
FETCH NEXT FROM cursor_tab1 INTO @id_tab1, @name
WHILE @@FETCH_STATUS = 0
BEGIN
   INSERT INTO tab2 (ID,firstname)
   VALUES (@id_tab1, @name)
FETCH NEXT FROM cursor_tab1 INTO @id_tab1, @name
END
CLOSE cursor_tab1
DEALLOCATE cursor_tab1
```

Výpis 42: Konkrétní příklad č.4

Tabulka 7: Tabulka spárovaných proměnných k příkladu č.4

| Atribut | Proměnná |
|---|---|
| ID | `@id_tab1` |
| firstname | `@name` |

Nutné podmínky, které musí být u nynějšího příkladu splněny, aby následující konverze do SQL byla korektní:

- U příkazu INSERT INTO za klauzulí VALUES můžou být přítomny jen konstanty, konstantní proměnné nebo vázané proměnné.

Řešení máme zachyceno ve výpisu (viz výpis 43).

```
INSERT INTO tab2(ID,firstname)
SELECT ID,firstname FROM tab1
WHERE EXISTS(
            SELECT ID,firstname
            FROM tab1
            )
```

Výpis 43: Řešení konkrétního příkladu č.4

### 3.10.6 Konkrétní příklad č.5

V tomto T-SQL kódu (viz výpis 44) procházíme všechny záznamy z tabulky „tab1“, abychom našli záznam, který má největší hodnotu pod atributem „value“.

```
DECLARE cursor_tab1 CURSOR FOR SELECT id,[name],[value] FROM tab1 Order by [value] desc
DECLARE @name as NVARCHAR(50);
DECLARE @id_tab1 as INT;
DECLARE @value as FLOAT;
DECLARE @max as FLOAT;
DECLARE @id as FLOAT;
```

```
OPEN cursor_tab1
FETCH NEXT FROM cursor_tab1 INTO @id_tab1, @name, @value
SET @max=@value
SET @id=@id_tab1
WHILE @@FETCH_STATUS = 0
BEGIN
   IF @value > @max
   BEGIN
      SET @max=@value
      SET @id=@id_tab1
   END
FETCH NEXT FROM cursor_tab1 INTO @id_tab1, @name, @value
END
SELECT id, [name], [value]
FROM tab1
Where id=@id
CLOSE cursor_tab1
DEALLOCATE cursor_tab1
```

Výpis 44: Konkrétní příklad č.5

Tabulka 8: Tabulka spárovaných proměnných k příkladu č.5

| **Atribut** | **Proměnná** |
|---|---|
| ID | `@id_tab1` |
| firstname | `@name` |
| value | `@value` |

Nutné podmínky, které musí být u nynějšího příkladu splněny, aby následující konverze do SQL byla korektní:

- Všechny příkazy SET se musí vyskytnout ve stejné logické podobě a v daném sledu, aby se nezměnila logika.
- U příkazu SELECT za klauzulí WHERE musí být přítomná podmínka s vázanou proměnnou.

Řešení je zachyceno ve výpisu 45.

```
SELECT top 1 id, name, value
FROM tab1
Order by value desc
```

Výpis 45: Řešení konkrétního příkladu č.5

### 3.10.7 Konkrétní příklad č.6

Tento příklad T-SQL kódu (viz výpis 46) nám zjistí, kolik se vyskytuje záznamů se jménem „karel“ v tabulce „tab1“.

```
DECLARE cursor_tab1 CURSOR FOR SELECT ID, [name] FROM tab1
DECLARE @pocet as INT
DECLARE @id_tab1 as INT
DECLARE @name as NVARCHAR(50);

OPEN cursor_tab1
FETCH NEXT FROM cursor_tab1 INTO @id_tab1, @name
SET @pocet=0
WHILE @@FETCH_STATUS = 0
BEGIN
   IF @name='karel'
   BEGIN
      SET @pocet = @pocet + 1
   END
FETCH NEXT FROM cursor_tab1 INTO @id_tab1, @name
END
SELECT 'karel' as namee, @pocet as pocet
CLOSE cursor_tab1
DEALLOCATE cursor_tab1
```

Výpis 46: Konkrétní příklad č.6

Nutné podmínky, které musí být u nynějšího příkladu splněny, aby následující konverze do SQL byla korektní:

- Všechny příkazy SET se musí vyskytnout ve stejné logické podobě a v daném sledu, aby se nezměnila logika.
- U příkazu SELECT za klauzulí WHERE musí být přítomná podmínka s vázanou proměnnou.

Tabulka 9: Tabulka spárovaných proměnných k příkladu č.6

| Atribut | Proměnná |
|---|---|
| ID | `@id_tab1` |
| firstname | `@name` |

Řešení je zachyceno ve výpisu 47.

```
SELEC [name] as jmeno,0+Count(*)*1 as pocet
FROM tab1
WHERE [name] = 'karel'
GROUP BY [name]
```

Výpis 47: Řešení konkrétního příkladu č.6

### 3.10.8 Konkrétní příklad č.7

V tomto T-SQL kódu (viz výpis 48) se ke každému záznamu v tabulce „tab2“ přičtou body. Body se přičítají podle toho, na jaké pozici se záznam nachází v tabulce „tab1“, která je seřazená dle ceny.

```
DECLARE cursor_tab1 CURSOR FOR SELECT ID, firstname FROM tab1 order by cena
DECLARE @id_tab1 as INT;
DECLARE @body as INT;
SET @body=0
OPEN cursor_tab1
FETCH NEXT FROM cursor_tab1 INTO @id_tab1, @name
WHILE @@FETCH_STATUS = 0
BEGIN
   UPDATE tab2
   SET body= body + @body
   WHERE ID=@id_tab1;
   SET @body= @body+1
   FETCH NEXT FROM cursor_tab1 INTO @id_tab1, @name
END
CLOSE cursor_tab1
DEALLOCATE cursor_tab1
```

Výpis 48: Konkrétní příklad č.7

Tabulka 10: Tabulka spárovaných proměnných k příkladu č.7

| **Atribut** | **Proměnná** |
|---|---|
| ID | `@id_tab1` |
| firstname | `@name` |

Nutné podmínky, které musí být u nynějšího příkladu splněny, aby následující konverze do SQL byla korektní:

- Všechny proměnné za příkazy SET se musí vyskytnout v daném sledu, aby se nezměnila logika.
- V deklaraci kurzoru u příkazu SELECT musí být přítomná klauzule ORDER BY.
- U příkazu UPDATE za klauzulí WHERE musí být přítomná podmínka s vázanou proměnnou.
- Jak jsme si popsali již v analýze výše, tak řešení uvedené níže je korektní, jen pokud atribut ID z tabulky „tab1“ bude unikátní (např. primární klíč).

Řešení je zachyceno na výpisu 49.

```
UPDATE gnt1
SET body = gnt1.body + gnt2.rn
FROM tab2 gnt1
JOIN (
      SELECT ID,firstname,0 + (ROW_NUMBER() OVER(order by cena) - 1) * 1 as rn
      FROM tab1
) gnt2 ON gnt1.ID = gnt2.ID
WHERE EXISTS(
            SELECT ID, firstname
            FROM tab1 order by cena)
```

Výpis 49: Řešení konkrétního příkladu č.7

### 3.10.9 Konkrétní příklad č.8

V tomto T-SQL kódu (viz výpis 50) se nám aktualizují záznamy z tabulky „tab2“ a to u atributu „body“, ke kterému se přičítá buďto jeden nebo dva body v závislosti na tom, jestli se shodují hodnoty atributu „firstname“ s předešlým záznamem z tabulky „tab1“ seřazené dle ceny, či nikoliv.

```
DECLARE cursor_tab1 CURSOR FOR SELECT ID,firstname FROM tab1 order by cena
DECLARE @name as NVARCHAR(50);
DECLARE @id_tab1 as INT;
DECLARE @lastname as NVARCHAR(50);

SET @lastname='karel'
OPEN cursor_tab1
FETCH NEXT FROM cursor_tab1 INTO @id_tab1, @name
WHILE @@FETCH_STATUS = 0
BEGIN
   If  @lastname=@name
      UPDATE tab2
      SET body=body+1
      WHERE ID=@id_tab1
   ELSE
      UPDATE tab2
      SET body=body+2
      WHERE ID=@id_tab1

   SET @lastname = @name
   FETCH NEXT FROM cursor_tab1 INTO @id_tab1, @name
END
CLOSE cursor_tab1
DEALLOCATE cursor_tab1
```

Výpis 50: Konkrétní příklad č.8

Tabulka 11: Tabulka spárovaných proměnných k příkladu č.8

| Atribut | Proměnná |
|---|---|
| ID | `@id_tab1` |
| firstname | `@name` |

Nutné podmínky, které musí být u nynějšího příkladu splněny, aby následující konverze do SQL byla korektní:

- Všechny proměnné za příkazy SET se musí vyskytnout v daném sledu, aby se nezměnila logika.
- V deklaraci kurzoru u příkazu SELECT musí být přítomná klauzule ORDER BY.

- U příkazu UPDATE za klauzulí WHERE musí být přítomná podmínka s vázanou proměnnou.
- Jak jsme si popsali již v analýze výše, tak řešení uvedené níže (viz výpis 51) je korektní, jen pokud atribut „ID“ z tabulky „tab1“ bude unikátní (např. primární klíč).
- Kvůli použité funkci LAG musí být atribut „firstname“ z tabulky „tab1“ NOT NULL, jinak následující řešení nebude zcela korektní.

Řešení je zachyceno ve výpisu 51.

```
UPDATE gnt1
SET body = gnt1.body + 1
FROM tab2 gnt1
join
   (
         Select *
         From (
               Select ID,firstname,LAG(firstname,1,'karel') OVER(ORDER BY cena) as previous
               From tab1
               ) s
         Where firstname = previous
   ) gnt2 ON gnt2.ID = gnt1.ID
WHERE EXISTS(
            SELECT ID,firstname
            FROM tab1 order by cena
            )

UPDATE gnt1
SET body = gnt1.body + 2
FROM tab2 gnt1
join
   (
      Select *
      From (
            Select ID,firstname,LAG(firstname,1,'karel') OVER(ORDER BY cena) as previous
            From tab1
            ) s
      Where firstname != previous
   ) gnt2 ON gnt2.ID = gnt1.ID
WHERE EXISTS(
            SELECT ID,firstname
            FROM tab1 order by cena)
```

Výpis 51: Řešení konkrétního příkladu č.8

# 4 Program

Součástí této bakalářské práce bylo i vytvoření programu, který umí převést konkrétní příklady uvedené v bakalářské práci. Program byl vytvořen jako desktopová aplikace pro operační systém Microsoft Windows. Aplikace byla napsána pomocí Visual Studia 2017. Při vytváření aplikace byl použit programovací jazyk `C#` a grafické uživatelské rozhraní bylo vytvořeno skrze WinForms. Hlavní funkcí této desktopové aplikace je převod T-SQL kódu s kurzorem na T-SQL kód bez kurzoru. Pro zpracování T-SQL kódu v programu byla použita knihovna třetí strany, a to knihovna pro parsování SQL kódu, přesnější „general SQL parser“ (A.2).

## 4.1 Popis aplikace

Aplikace obsahuje mnoho tříd, ale nás budou zajímat především třídy Optimizer, Analyzer a Converter.

### 4.1.1 Optimizer

Tato třída se jeví jako černá skříňka, do které pošlete T-SQL kód s kurzorem a nazpátek dostanete SQL kód. Ve skutečnosti tato třída v sobě obsahuje třídu Analyzer a Converter, které se o úpravu kódu postarají.

### 4.1.2 Analyzer

Tato třída se stará o analýzu T-SQL kódu od uživatele. Analyzer kód zpracuje a dá se říct, že doslova nakrmí potřebnými informacemi třídu Converter, která se stará o následnou konverzi kódu. Analyzer se také stará o šablony a jejich porovnávání se vstupním T-SQL kódem.

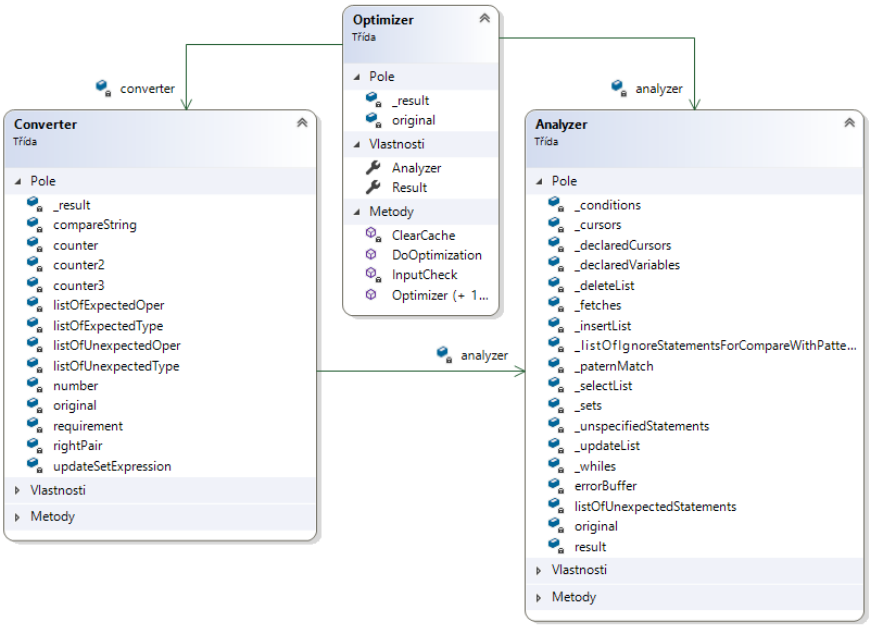
#### 4.1.2.1 Šablony

Šablony nám zde slouží k porovnání syntaxe. Přesněji se každá šablona porovná s T-SQL kódem od uživatele tak, že se porovnává příkaz po příkazu a když má kód stejnou strukturu, je nalezena shoda a na základě informace, o kterou šablonu se jedná, se aplikace rozhodne, jak bude postupovat dále.

Nyní jsou šablony tvořeny pouze funkčním T-SQL kódem a číslem pro metodu, která se postará o následnou konverzi. V budoucnu by bylo dobré šablony zobecnit, aby se s nimi lépe pracovalo. Pro další vylepšení se do šablon může zapracovat sémantika neboli logika, která se nyní řeší až v třídě Converter.

### 4.1.3 Converter

Tato třída se stará o konverzi T-SQL kódu. Converter de facto neobsahuje nic jiného než metody pro úpravu kódu. Na základně dat ze třídy Analyzer se rozhodne o tom, jakou metodu ke konverzi




Obrázek 7: Zjednodušený diagram tříd

T-SQL kódu použije a následně ji vykoná. Výsledek konverze si následně vyzvedne Optimizer a předá jej uživatelskému rozhraní, pro zobrazení uživateli.

## 4.2 Popis funkčnosti programu

Nejprve si zhruba popíšeme, jak aplikace funguje a poté se podíváme na nějaké detaily.

Poté, co aplikace obdrží SQL kód od uživatele, začne analýza kódu. V této fázi si aplikace projde všechny příkazy, které se v kódu vyskytují a každý z nich si nějakým způsobem předpřipraví pro následující transformaci. Pro transformaci T-SQL kódu se v aplikaci využívají šablony. Máme zde šablony dvojího typu. První typ šablon se využívá k porovnání celého kódu najednou. Tento typ šablon zde máme proto, abychom dokázali řešit i případy, kde tělo cyklu kurzoru ovlivňuje i okolní příkazy. Druhý typ šablon se porovnává pouze s tělem cyklu kurzoru. Základem aplikace je cyklus, který provádí T-SQL transformace dokud existuje šablona nebo optimalizační metoda, kterou je možné na kód aplikovat. V každé iteraci se aplikace pokouší transformovat T-SQL kód. Šablony jsou v aplikaci uloženy v XML souborech. Příklad šablony z XML máme ve výpisu 52.

```
<Pattern>
    <ConvertMethod>1</ConvertMethod>
    <Code>
  WHILE @@FETCH_STATUS = 0
  BEGIN
        DELETE FROM [Customer]
         WHERE ID = @id_tab1
        FETCH NEXT FROM cursor_tab1
        INTO @id_tab1, @name
```

```
END
</Code>
</Pattern>
```

Výpis 52: šablona v XML

Porovnání šablony s kódem se v aplikaci provádí podobně jako průchod stromem do hloubky, kdy aplikace musí procházet zároveň kód od uživatele a kód šablony. K tomuto účelu byl velice nápomocný „general SQL parser“ (A.2), který příkazy naparsoval. Aplikace tedy pro porovnání šablon postupuje následovně.

SQL parser rozparsuje příkazy do struktury derivačního stromu, tento strom se porovná se stromem příkazů ze šablony. Jestli se všechny uzly stromů vzájemně shodují, tak je nalezena shoda se šablonou a podle atributu „NumberMethod“, který je uložený v XML společně se šablonou, se rozhodne, která metoda se použije v třídě Converter k následné transformaci kódu.

Aplikace konkrétně postupuje následovně. Nejdříve se porovná vstupní kód se šablonami prvního typu (šablony pro celý T-SQL kód). Jestli není nalezena shoda, tak se ze vstupního kódu porovnává cyklus kurzoru (jestli je kurzorů víc, tak se začne s tím, co je nejvíce vnořený) se šablonami druhého typu (šablony pro cyklus kurzoru). Jestli je nalezena shoda, tak se daný cyklus kurzoru nahradí SQL dotazem a následně se jen udělají úpravy „pro úklid“. Odstraní se nadbytečný FETCH příkaz, pokud není kurzor používán v jiném cyklu kurzoru, tak se odstraní i příkazy OPEN, DEALLOCATE, CLOSE pro daný kurzor a také definice kurzoru.

Aplikace umí transformovat T-SQL kód ze všech konkrétních příkladů uvedných v sekci 3.10. Dále si poradí s transformací kurzoru pro příkaz DELETE a to pro všechny případy zachycené v analýze, i pro ty kdy se v těle kurzoru objeví podmínka IF. Ostatní JMD příkazy a jejich kombinace jsou otázkou budoucí implementace.

**Algoritmus 1:** Pseudokód pro porovnání vstupu se šablonou

```
XML ← load XML from XML file;
Patterns ← Convert XML to Patterns;
patternCount ← count of Patterns;
counter ← 0;
while counter is equal to patternCount do
    if compare(input, Patterns(counter)) then
        return GetConvertMethod(pattern);
    end if
    counter ← counter+1;
end while
```

## 4.3 Uživatelské rozhraní

Uživatelské rozhraní má čtyři hlavní okna. Tato okna jsou „Vstup“, “Nastavení“, “Šablona“ a „Výstup“.

**Algoritmus 2:** Pseudokód aplikace

```
input ← code from user;
while input is equal to output do
    input ← output;
    analyze ← Analyzes(input);
    pattern ← FindPattern(input);
    output ← DoConvertion(analyze, pattern);
end while
return output;
```

### 4.3.1 Vstup

V tomto okně se nachází textbox, do kterého můžeme zapsat vstup nebo pomocí tlačítka „načti“ jej načteme ze souboru. U textu, který jsme vložili, se nám po dvou vteřinách zvýrazní klíčová slova. Dále zde máme tlačítko „Optimalizuj“. Když klikneme na tlačítko „Optimalizuj“, tak nás aplikace přesune do okna výstup a v tomto okně se nám zobrazí přeložený kód bez kurzoru.


Obrázek 8: Ukázka programu

### 4.3.2 Nastavení

V okně nastavení si můžeme zvolit, které příkazy se nám nebudou brát v potaz při porovnávání se šablonou.

### 4.3.3 Šablona

V okně šablony si můžeme upravit, přidat, či odebrat šablonu.

### 4.3.4 Výstup

V okně výstupu se nám zobrazí přeložený kód bez kurzoru. Tento kód si následně můžeme uložit do souboru.

# 5 Závěr

Cílem této bakalářské práce bylo udělat analýzu T-SQL kódu s kurzorem a vytvořit nástroj, který bude umět přeložit T-SQL kód na kód bez kurzoru. V této bakalářské práci se mi podařilo popsat mnoho JMD příkladů v T-SQL kódu s kurzorem. U každého příkladu jsem uvedl postup i řešení k danému převodu. Pokud řešení mělo nějaké omezení, tak je zde také popsáno. V sekci pro analýzu T-SQL kódu jsem se převážně věnoval JMD příkazu DELETE, u něhož jsem popsal mnoho situací, které se v kurzoru mohou objevit. Dále jsem v analýze uvedl odlišnosti u JMD příkazů UPDATE a INSERT INTO v kontextu spjatém s T-SQL kurzorem. V dokumentu jsem také zachytil 8 konkrétních příkladů, u nichž jsem uvedl řešení a podmínky, za kterých je řešení korektní. Dále je součástí této bakalářské práce desktopová aplikace, která umí převést uvedené konkrétní příklady T-SQL kódu na SQL. Tato desktopová aplikace byla vytvořena jen pro studijní účely této bakalářské práce.

V budoucnu by jistě stálo zato, podrobně zanalyzovat závislost mnohonásobného výskytu JMD příkazů v těle cyklu kurzoru a podmínky OR pro všechny JMD příkazy. Dále by se měly zanalyzovat závislosti spojené s proměnnou set, což bude podle mě velice obtížné nějak korektně zachytit.

Pro vylepšení desktopové aplikace by do budoucna bylo dobré zaimplementovat všechna pravidla, která jsou zachycena v příkladech analýzy.

# Literatura


[1] ROUSE, Margaret a Jessica SIRKIN. SQL (Structured Query Language) [online]. září 2016 [cit. 2018-06-25]. Dostupné z: https://searchsqlserver.techtarget.com/definition/SQL

[2] ROUSE, Margaret, Sten JONES a Adam HUGHES. T-SQL (Transact-SQL) [online]. červen 2017 [cit. 2018-06-25]. Dostupné z: https://searchsqlserver.techtarget.com/definition/SQL

[3] LAUDENSCHLAGER, Douglas, Craig GUYER a Caro CASERIO. DECLARE CURSOR (Transact-SQL) [online]. 14. března 2017 [cit. 2018-06-25]. Dostupné z: https://docs.microsoft.com/en-us/sql/t-sql/language-elements/declare-cursor-transact-sql?view=sql-server-2017

[4] LAUDENSCHLAGER, Douglas a Craig GUYER. FETCH (Transact-SQL) [online]. 14. března 2017 [cit. 2018-06-25]. Dostupné z: https://docs.microsoft.com/en-us/sql/t-sql/language-elements/fetch-transact-sql?view=sql-server-2017

[5] LAUDENSCHLAGER, Douglas a Craig GUYER. CLOSE (Transact-SQL) [online]. 6. března 2017 [cit. 2018-06-25]. Dostupné z: https://docs.microsoft.com/en-us/sql/t-sql/language-elements/close-transact-sql?view=sql-server-2017

[6] LAUDENSCHLAGER, Douglas a Craig GUYER. OPEN (Transact-SQL) [online]. 14. března 2017 [cit. 2018-06-25]. Dostupné z: https://docs.microsoft.com/en-us/sql/t-sql/language-elements/open-transact-sql?view=sql-server-2017

[7] LAUDENSCHLAGER, Douglas a Craig GUYER. DEALLOCATE (Transact-SQL) [online]. 14. března 2017 [cit. 2018-06-25]. Dostupné z: https://docs.microsoft.com/en-us/sql/t-sql/language-elements/deallocate-transact-sql?view=sql-server-2017

# A Přílohy

## A.1 Obsah CD

- bp - Zde je PDF verze dokumentu bakalářské práce
- program - Zde je uložen spustitělný program
- src - Tato složka obsahuje zdrojové kódy programu

## A.2 Použité knihovny třetích stran

- general SQL parser - http://www.sqlparser.com/

## A.3 Zmíněné nástroje

- Qure Optimizer od DBSophic - http://www.dbsophic.com/